

**METHOD AND SYSTEM IN AN ELECTRONIC SPREADSHEET FOR PERSISTENTLY  
SELF-REPLICATING MULTIPLE RANGES OF CELLS THROUGH A COPY-PASTE  
OPERATION**

***Field of the invention***

5 The present invention relates to the field of information processing by digital computers, and more particularly to a method and system, in an electronic spreadsheet, for persistently self-replicating multiple ranges of cells through a copy-paste operation.

***Background***

10 Before computers, numerical analyses, particularly financial ones, were usually prepared on an accountant's columnar pad or spreadsheet, with pencil and calculator in hand. By organizing data into columns and rows, spreadsheets afford the rapid  
15 assimilation of information by a reader. The task of preparing a spreadsheet on paper, however, is laborious. The process tends to be very slow, as each entry must be tediously calculated and entered into the spreadsheet. Since all calculations are the responsibility of the preparer, manually prepared spreadsheets  
20 are also prone to errors. Hence, preparation of spreadsheets by hand is slow, tedious, and unreliable.

25 With the advent of microcomputers, a solution was forthcoming in the form of "electronic spreadsheets." Better known simply as "spreadsheets," these software programs provide a computerised replacement for the traditional financial modelling tools: the

accountant's columnar pad, pencil, and calculator. In some regards, spreadsheet programs are to those tools what word processors are to typewriters. Spreadsheets offer dramatic improvements in ease of creating, editing, and using financial models.

A typical spreadsheet program configures the memory of a computer to resemble the column/row or grid format of an accountant's columnar pad, thus providing a visible calculator for a user. Because this "pad" exists dynamically in the computer's memory, however, it differs from paper pads in several important ways. Locations in the electronic spreadsheet, for example, must be communicated to the computer in a format which it can understand. A common scheme for accomplishing this is to assign a number to each row in a spreadsheet, a letter to each column, and another letter to each sheet (or page) of the spreadsheet. To reference a location at column A and row 1 of the second page (i.e., the upper-left hand corner), for example, the user types in "B:A1". In this manner, the spreadsheet defines an addressable storage location or "cell" at each intersection of a row with a column within a given page.

Data entry into an electronic spreadsheet occurs in much the same manner that information would be entered on an accountant's pad. After a screen cursor is positioned at a desired location, the user can enter alphanumeric information. Besides holding text and numeric information, however, spreadsheet cells can store special instructions or "formulas" specifying calculations to be performed on the numbers stored in spreadsheet cells. Such spreadsheet cells can also be defined and named as a range as long as they are arranged as a contiguous set of cells. A typical example of such a named range simply corresponds to a regular table found in an accountant's pad. In this fashion, range names

can serve as variables in an equation, thereby allowing precise mathematical relationships to be defined between cells. The structure and operation of a spreadsheet program, including advanced functions such as functions and macros, are documented in the technical, trade, and patent literature.

Electronic spreadsheets offer many advantages over their paper counterparts. For one, electronic spreadsheets are much larger (i.e., hold more information) than their paper counterparts; electronic spreadsheets having thousands or even millions of cells are not uncommon. Spreadsheet programs also allow users to perform "what-if" scenarios. After a set of computational relationships has been entered into a worksheet, thanks to imbedded formulas for instance, the spread of information can be recalculated using different sets of assumptions, with the results of each recalculation appearing almost instantaneously. Performing this operation manually, with paper and pencil, would require recalculating every relationship in the model with each change made. Thus, electronic spreadsheet systems were invented to solve "what-if" problems, that is, changing an input and seeing what happens to an output.

Cell ranges are used to automate the computations in a spreadsheet. Whether cells or cell ranges are named or not, they can be referenced within a formula either by a "relative" or an "absolute" reference. Such a reference can be the address of the referenced cell range, or the name of the referenced cell range if it turns that this cell range is named.

It is common to find in electronic spreadsheet based applications some large tables which are organised according to a structured way. This structure typically results in organising rows, columns and sheets in such a way that the content of each of the cells

within a given column and within a given sheet can be obtained as the result of a copy-paste operation where the source copied cell is any cell within this same column and same sheet. In such typical situations, this source cell can contain a formula

5 referencing in a relative or absolute way one or several other cells, so that each of the other cells within the same column of the same sheet will also contain the same formula where the absolute references will be kept unchanged and where the relative references will point to other relative cells.

10 Such a typical situation is illustrated in FIG 3A where a table is used to compute a sales item price according to some input data. In this table, the content of the cell with address C6 (column entitled "Unit Cost") is for instance equal to the formula "@CostOf(B6)" where @CostOf is a dedicated function

15 providing the cost of an item passed as parameter. In the same table, the content of the cell with address G6 (column entitled "Exchange rate") is for instance equal to the formula "@RateOf(F6)" where @RateOf is a dedicated function returning the exchange rate for a currency passed as parameter. In the same

20 table, the content of the cell with address I6 (column entitled "Price") is for instance equal to the formula "C6\*D6\*G6/(1-\$PROFIT)" where "PROFIT" is the name given to the cell range with address I3 where the profit figure is recorded. The content of each cell within the "Unit Cost" table can be

25 obtained by copy-pasting the cell with address C6, so that the content of the cell with address Cx (where x takes the values 7 to 10) is found equal to "@CostOf(Bx)". In this way, each of the cells with address C6 to C10 is virtually a "replicate" of all the other cells with address C6 to C10 through a copy-paste

30 operation, meaning that any cell within this set can be derived from any other one within the same set through a copy-paste operation. Similarly, the content of the cells with address Gx

and with address Ix are obtained by copy-pasting the content of the cells with address G6 and with address I6, respectively. The content is equal to "@RateOf(Fx)" and to "Cx\*Dx\*Gx/(1-\$PROFIT)" respectively. Thus, the cells with address G6 to G10 and the cells with address I6 to I10 are virtually "self replicating" through a copy-paste operation. The copy-paste operation is thus a powerful tool for applying in many different cells, or ranges of cells, the content of a given cell or of a given range of cells. Nevertheless this copy-paste operation presents some limitations, as outlined hereafter.

Assume that in our example the content of the cells within a table column needs to be updated to reflect some structural change of the table it belongs to. Such a structural change is illustrated in FIG 3B where the profit parameter (used to derive a price from a cost) is no longer constant for all sold items (as shown in FIG 3A with the cell of address I3, and named "PROFIT"), but depends on the sold item itself, as represented in the table by the cells within the column entitled "Profit". Under this new rule, the content of the cell with address I6 (within the column entitled "Price") is now equal to the formula "C6\*D6\*G6/(1-H6)". In order to reflect this table structural update in the other cells of the same "Price" column, it is necessary to reapply the copy-paste operation from the top column cell to all the other column cells following the same logic, that is the cells with address I7 to I10 as shown in FIG 3B. More generally, this operation must be carefully done each time a given range of cells content is updated and must be applied to all the other ranges of cells which have been initially self replicated with this given range of cells through a copy-paste operation.

With large and complex spreadsheets, such a task may take quite a long time and be error prone, because the spreadsheet user may

miss some of the ranges of cells where the copy-paste operation must be reapplied. When this happens, the resulting spreadsheet provides erroneous results. The present invention offer a powerful and efficient solution to this problem by defining a method and a system for persistently self-replicating multiple ranges of cells through a copy-paste operation.

### ***Summary of the invention***

The present invention relates to the field of information processing by digital computers, and more particularly to a method and system for persistently self-replicating multiple ranges of cells through a copy-paste operation, in a multi dimensional spreadsheet. The method comprises the steps of:

- defining a set of ranges of cells, each range of cells having the same size;
- each time the content of a range of cells belonging to this set is changed, automatically performing a self-replication operation, the self-replication operation comprising the steps of:
  - copying the changed range of cells onto a buffer;
  - determining the set of ranges of cells to which the changed range of cells belongs to;
  - identifying the ranges of cells belonging to the set; and
  - pasting the content of the buffer in each identified range of cells belonging to the set.

### ***Brief description of the drawings***

The invention will best be understood by reference to the following detailed description when read in conjunction with the accompanying drawings, wherein::

- Figure **1A** is a schematic view of a computer system in which the present invention may be embodied.
- Figure **1B** is a schematic view a software system including an operating system, software application, and a user interface for carrying out the present invention.
- Figure **1C** illustrates the basic architecture and functionality of a graphical user interface in which the present invention may be embodied.
- Figure **2A** shows a spreadsheet notebook interface according to a preferred embodiment of the present invention.
- Figure **2B** shows the toolbar component of the notebook interface shown in Figure **2A**.
- Figures **2C** and **2D** show page identifiers for rapidly accessing and manipulating individual pages of the notebook interface shown in Figure **2A**.
- Figures **3A** and **3B** illustrate a typical spreadsheet structure used in a preferred embodiment of the present invention.
- Figure **4** illustrates the structure of the persistent self-replication table, according to a preferred embodiment of the present invention.

- Figures **5A**, **5B**, illustrate a spreadsheet user interface for invoking the persistent self-replicating operation, according to the present invention.
- Figures **6A**, **6B** are a flow chart illustrating a method for managing objects involved in Persistent Self-Replication operations, according to the present invention.
- Figure **7** is a flow chart illustrating a method for performing a persistent copy-paste operation, according to the present invention.

### *Detailed description*

#### **SYSTEM HARDWARE**

As shown in FIG. **1A**, the present invention may be embodied on a computer system **100** comprising a central processor **101**, a main memory **102**, an input/output controller **103**, a keyboard **104**, a pointing device **105** (e.g., mouse, track ball, pen device, or the like), a display device **106**, and a mass storage **107** (e.g., hard disk). Additional input/output devices, such as a printing device **108**, may be included in the system **100** as desired. As illustrated, the various components of the system **100** communicate through a system bus **110** or similar architecture. In a preferred embodiment, the computer system **100** includes an IBM-compatible personal computer, which is available from several vendors (including International Business Machine - IBM Corporation of Armonk, N.Y.).



Illustrated in FIG. 1B, a computer software system 150 is provided for directing the operation of the computer system 100. Software system 150, which is stored in system memory 102 and on disk memory 107, includes a kernel or operating system 151 and a shell or interface 153. One or more application programs, such as application software 152, may be "loaded" (i.e., transferred from storage 107 into memory 102) for execution by the system 100. The system 100 receives user commands and data through user interface 153; these inputs may then be acted upon by the system 100 in accordance with instructions from operating system 151 and/or application software 152. The interface 153, which is preferably a graphical user interface (GUI), also serves to display results, whereupon the user may supply additional inputs or terminate the session. In a preferred embodiment, operating system 151 and interface 153 are Microsoft Win95, available from Microsoft Corporation of Redmond, Wash. Application software 152, on the other hand, includes a spreadsheet notebook of the present invention as described in further detail herein below.

## **INTERFACE**

### **A. Introduction**

The following description will focus on embodiments of the present invention that include spreadsheet applications operative in the Microsoft Win95 environment. The present invention, however, is not limited to any particular application or any particular environment. Instead, those skilled in the art will find that the system and methods of the present invention may be advantageously applied to a variety of system and application software, including database management systems, word processors, and the like. Moreover, the present invention may be embodied on

a variety of different platforms, including Macintosh, UNIX, NextStep, and the like. Therefore, the descriptions of the exemplary embodiments which follows are for purposes of illustration and not limitation.

Referring now to FIG. 1C, the system 100 includes a workspace or window 160. Window 160 is a rectangular, graphical user interface (GUI) for display on screen 106; additional windows may be displayed in various sizes and formats (e.g., tiled or cascaded), as desired. At the top of window 160 is a menu bar 170 with a plurality of user-command choices, each of which may invoke additional submenus and software tools for use with application objects. Window 160 includes a client area 180 for displaying and manipulating screen objects, such as graphic object 181 and text object 182. In essence, the client area is a workspace or viewport for the user to interact with data objects which reside within the computer system 100.

Window 160 includes a screen cursor or pointer 185 for selecting and otherwise invoking screen objects of interest. In response to user movement signals from the pointing device 105, the cursor 185 floats (i.e., freely moves) across the screen 106 to a desired screen location. During or after cursor movement, the user may generate user-event signals (e.g., mouse button "clicks" and "drags") for selecting and manipulating objects, as is known in the art. For example, window 160 may be closed, re-sized, or scrolled by "clicking" (selecting) screen components 172, 174/5, and 177/8, respectively.

In a preferred embodiment, screen cursor 185 is controlled with a mouse device. Single-button, double-button, or triple-button mouse devices are available from a variety of vendors, including

Apple Computer of Cupertino, Calif., Microsoft Corporation of Redmond, Wash., and Logitech Corporation of Fremont, Calif., respectively. More preferably, screen cursor control device **105** is a two-button mouse device, including both right and left "mouse buttons."

Programming techniques and operations for mouse devices are well documented in the programming and hardware literature; see e.g., *Microsoft Mouse Programmer's Reference*, Microsoft Press, 1989. The general construction and operation of a GUI event-driven system, such as Microsoft Windows, is also known in the art: see, e.g., Petzold, C., *Programming Windows*, Second Edition, Microsoft Press, 1990. The disclosures of each are hereby incorporated by reference.

#### **B. Preferred interface**

A spreadsheet notebook interface of the present invention will now be described. As shown in Fig. 2A, the spreadsheet notebook or workbook of the present invention includes a notebook workspace **200** for receiving, processing, and presenting information, including alphanumeric as well as graphic information. Notebook workspace **200** includes a menu bar **210**, a toolbar **220**, a current cell indicator **230**, an input line **231**, a status line **240**, and a notebook window **250**. The menu bar **210** displays and invokes, in response to user inputs, a main level of user commands. Menu **210** also invokes additional pull down menus, as is known in windowing applications. Input line **231** accepts user commands and information for the entry and editing of cell contents, which may include data, formulas, macros, and the like. Indicator **230** displays an address for the current cursor (i.e., active cell) position, or the address or name of a selected named range (i.e. active selection). At the status line **240**, system **100** displays information about the current state of the workbook; for

example, a "READY" indicator means that the system is ready for the user to select another task to be performed.

5 The toolbar **220**, shown in further detail in FIG. 2B, comprises a row or palette of tools which provide a quick way for the user to choose commonly-used menu commands or properties. In an exemplary embodiment, toolbar **220** includes file manipulation buttons **221**, printing buttons **222**, an undo button **223**, cut, copy, and paste buttons **224**, information pop-up window buttons tool **225**, a named range selection button **226**, a style copy button **227**, a column re-sizing button **228**, and a sum button **229**. The functions of these buttons are suggested by their names. For instance, buttons **224** cut, copy and paste data and objects to and from Windows' clipboard. The same actions are also available as corresponding commands in the Edit menu (available from menu bar **210**).

15 The notebook, which provides an interface for entering and displaying information of interest, includes a plurality of spreadsheet pages. Each page may include conventional windowing features and operations, such as moving, re-sizing, and deleting. In a preferred embodiment, the notebook includes 256 spreadsheet pages, all of which are saved as a single disk file on the mass storage **107**. Workspace **200** may display one or more notebooks, each sized and positioned (e.g., tiled, overlapping, and the like) according to user-specified constraints.

25 Each spreadsheet page of a notebook includes a two-dimensional spread. Page A from the notebook **200**, for example, includes a grid in row and column format, such as row 3 and column F. At each row/column intersection, a box or cell (e.g., cell **C4**) is provided for entering, processing, and displaying information in a conventional manner. Each cell is addressable, with a selector

being provided for indicating a currently active cell (i.e., the cell that is currently selected).

As shown in FIGS. **2C-D**, individual notebook pages are identified by page identifiers **260**, preferably located along one edge of a notebook. In a preferred embodiment, each page identifier is in the form of a tab member (e.g., members **261a**, **262a**, **263a**) situated along a top edge of the notebook. Each tab member may include representative indicia, such as textual or graphic labels, including user selected titles representing the contents of a corresponding page. In FIG. **2C**, the tab members **260** are set to their respective default names. For example, the first three tab members (members **261a**, **262a**, **263a**) are respectively set to A, B, and C. Tab members are typically given descriptive names provided by the user, however. As shown in FIG. **2D**, for example, the first three tab members have now been set to "Contents" (tab member **261b**), "Summary" (tab member **262b**), and "Jan" (tab member **263b**). In a similar manner, the remaining tabs are set to subsequent months of the year. In this manner, the user associates the page identifiers with familiar tabs from an ordinary paper notebook. Thus, the user already knows how to select a page or spread of interest: simply select the tab corresponding to the page (as one would do when selecting a page from a paper notebook).

In addition to aiding in the selection of an appropriate page of information, the user-customizable page identifiers serve to aid in the entry of spreadsheet named range addresses. For example, when entering a formula referring to a named range of cells on another page, the user may simply use the descriptive page name in the named range address, thus making it easier for the user to understand the relationship of the cell(s) or information being referenced.

A general description of the features and operation of the spreadsheet notebook interface may be found in Quattro Pro for Windows (*Getting Started, User's Guide and Building Spreadsheet Applications*), available from Borland International.

## 5 **PERSISTENT SELF-REPLICATING OPERATION**

### **A. Introduction**

As the power of spreadsheet environments has increased over the last several years, it is possible today to develop complex custom applications based solely on spreadsheets, as opposed to applications developed with general purpose programming languages like C++ or VisualBasic from Microsoft Corporation. This can be achieved using spreadsheet imbedded tools such as macro languages, script languages, and formulas. In large spreadsheets, it is common to find structured tables where the content of some cells are directly derived from the contents of other cells using formulas which translate the relationship between these cells.

Such formulas can be quite complex, consequently it is advantageous to copy-paste such a formula, once established, from a given cell onto all the other cells where the same relationship exists. If this relationship evolves during the life of a spreadsheet, however, then the spreadsheet user must first update a first cell content (typically rewriting the formula it holds), and then the spreadsheet user must again perform a copy-paste operation between this first cell and all the cells whose content was previously obtained from the reference cell content with a copy-paste operation. In addition to the time spent by the user in performing this operation, there is a risk of applying this new copy-paste operation to the wrong set of cells: either some cells that need to be copy-pasted again may be missed, or some

cells may be copy-pasted that should not be. In both cases, the resulting spreadsheet has been incorrectly updated, so that it provides erroneous results.

The present invention offers a user-friendly solution to this problem by defining a method and system that enables persistent self-replication of multiple ranges of cells through a copy-paste operation.

### **B. Self-Replication Manager**

In contrast to the conventional tools just-described, the present invention provides a more powerful, user-friendly and interactive approach for persistently self-replicating multiple ranges of cells through a copy-paste operation using a Self-Replication manager. The manager automatically allows the spreadsheet user to:

- create or rename or delete a set of persistently self-replicating ranges of cells, or
- add or suppress a given range of cells to or from a set of persistently self-replicating ranges of cells, and
- self-replicate any content update from a given range of cells belonging to a set of persistently self-replicating ranges of cells to all the other ranges of cells belonging to the same set of persistently self-replicating ranges of cells.

For descriptive clarity, a persistently self-replicating range of cells which can take advantage of the present invention will be called "*persistently self-replicating range*" or "*PSRR*", and a set of PSRR will be called "*persistently self-replicating set*" or "*PSRS*"

### **C. Persistently self-replicating range and persistently self-replicating set**

In a preferred embodiment, PSRR can be easily identified on the display device **106** within the work area **180** of the window **160** by using some specific cell attributes, such as a font style or font color or background color or border line style or border line color or background pattern, etc. In a preferred embodiment, the background pattern of a PSRR is set to a first pattern value referred to as PATTERN\_PSRR. PSRS are uniquely identified by a logical identifier which can take multiple forms. In a preferred embodiment of the present invention, PSRS are uniquely identified by a name corresponding to a character string of limited length.

#### D. Scenario

In a preferred embodiment, the present invention may be used in two steps:

- The first step occurs when the spreadsheet user decides, based on criteria not developed here, to take advantage of the present invention by using the self-replication manager for either creating, or deleting, or renaming a PSRS ; or for either adding or removing a PSRR to / from a PSRS.

If so, the spreadsheet user can follow in sequence the following steps :

- First the spreadsheet user optionally selects a range of cells by using conventional means such as (but not limited to) the pointing device **105** or the keyboard **104**.
- Then the spreadsheet user invokes an extension of the regular spreadsheet editing facilities using conventional means available in spreadsheet environment, such as (but not limited to) dedicated push-buttons, keyboard entry short cuts, menu or sub menu entries. This extension of the regular editing facilities corresponds to a specific command called "**Self\_Replication\_Manager**". In a preferred embodiment of the present invention, this **Self\_Replication\_Manager**



command is invoked by clicking with the pointing device **105** on a menu entry **501** "Self Replication" within the conventional "Edit" menu **500** of an electronic spreadsheet, as shown in FIG **5A**. It results in displaying on the display device **106** a specific Self-Replication Dialog Box **600**, as shown in FIG **5B**.

- Then the spreadsheet user can take advantage of the different tools available within the Self-Replication Dialog Box **600**, according to the following list:

- The "Name" text box **511** and the "Create" push-button **517** can be used to create a new PSRS. For this purpose the spreadsheet user first fills this "Name" text box **511** with the name of the new PSRS to be created and then clicks on the "Create" push-button **517**. As a result, the newly created PSRS now appears within the "Existing PSRS" list box **512**.

- The "Existing PSRS" list box **512** and the "Delete" push-button **518** can be used to delete an existing PSRS. For this purpose, the spreadsheet user first selects within the "Existing PSRS" list box **512** the name of the PSRS to be deleted (if not visible within the "Existing PSRS" list box **512**, the spreadsheet user can simply use the scroll bar **523** with the pointing device **105** to let the desired PSRS name appear within the "Existing PSRS" list box **512**), and then clicks on the "Delete" push-button **518**. As a result, the just deleted PSRS disappears from the "Existing PSRS" list box **512**.

- The "Name" text box **511**, the "Existing PSRS" list box **512** and the "Rename" push-button **519** can be used to change the name of an existing PSRS. For this purpose, the spreadsheet user first selects within the "Existing PSRS" list box **512** the name of the PSRS to be renamed (if not visible within the "Existing PSRS" list box **512**, the spreadsheet user can simply

use the scroll bar **523** with the pointing device **105** to let the desired PSRS name appear within the "Existing PSRS" list box **512**), then fills the "Name" text box **511** with the new name of the PSRS, and then clicks on the "Rename" push-button **518**. As a result, the new name of the existing PSRS now appears within the "Existing PSRS" list box **512**.

- The "Existing PSRR members" list box **513** can be used to visualize all the PSRRS belonging to the PSRS currently selected within the "Existing PSRS" list box **512**. To do this, the spreadsheet user first selects within the "Existing PSRS" list box **512** the name of the PSRS to be visualized (if not visible within the "Existing PSRS" list box **512**, the spreadsheet user can simply use the scroll bar **523** with the pointing device **105** to let the desired PSRS name appear within the "Existing PSRS" list box **512**), and then clicks on the scroll bar **524** with the pointing device **105** to display in the "Existing PSRR members" list box **513** every PSRR belonging to the PSRS selected in the "Existing PSRS" list box **512**.

- The "Existing PSRR members" list box **513** and the "Suppress" push-button **521** can be used to remove a given PSRR from the PSRS currently selected within the "Existing PSRS" list box **512**. To do this, the spreadsheet user first selects within the "Existing PSRS" list box **512** the name of the PSRS from which one member must be removed (if not visible within the "Existing PSRS" list box **512**, the spreadsheet user can simply use the scroll bar **523** with the pointing device **105** to let the desired PSRS name appear within the "Existing PSRS" list box **512**). The user then selects within the "Existing PSRR members" list box **513** the name of the PSRR to be removed (if not visible within the "Existing PSRR members" list box **513**, the spreadsheet user can simply use the scroll bar **524** with the pointing device **105** to let the desired PSRR name appear

within the "Existing PSRR members" list box **513**), and then clicks on the "Suppress" push-button **521**. As a result, the just deleted PSRR disappears from the "Existing PSRR" list box **513**, and its background pattern is changed from the value PATTERN\_PSRR to its original value (before it was turned as a PSRR).

- The "Range" text box **514**, the "Select" push-button **522** and the "Add" push-button **520** can be used to add a new PSRR to the PSRS currently selected within the "Existing PSRS" list box **512**. To do this, the spreadsheet user first checks that the "Range" text box **514** holds the address of the range of cells to be added. By default, the "Range" text box **514** contains the address of the range of cells which was selected in the electronic spreadsheet just before invoking the ***Self\_Replication\_Manager*** command. The user can change this default range by clicking on the "Select" push-button **522** and then use the pointing device **105** to select the desired range of cells within the electronic spreadsheet. When the "Range" text box **514** holds the address of the right range of cells to be added, the user clicks on the "Add" push-button **520**. As a result, the newly added PSRR now appears within the "Existing PSRR members" list box **513**, and its background pattern is changed into a new pattern with value PATTERN\_PSRR.

- The "Cancel" push-button **516** and the "OK" push-button **515** can be used by the spreadsheet user to close the Self-Replication Dialog Box **600**.

- The second step occurs when the spreadsheet user updates a cell belonging to a PSRR which is itself a member of a PSRS:

- If the spreadsheet user updates the content of a cell belonging to a PSRR, the self-replication manager invokes

by itself a specific command called  
**"Persistent\_Self\_Replicate"** which automatically reflects  
 this update in all the other PSRR belonging to the same  
 PSRS than the updated PSRR. This  
 5 **"Persistent\_Self\_Replicate"** operation is fully automated,  
 without involvement of the spreadsheet user, and is itself  
 based on a copy-paste operation applied by the  
 self-replication manager between the updated PSRR and all  
 the other PSRR belonging to the same PSRS.

#### 10 **E. Persistent Self-Replication Table**

The decision to create, delete, or rename a PSRS, or to add or  
 suppress a PSRR to or from a PSRS, belongs to the spreadsheet  
 user. When such an operation occurs, a common repository, called  
 the *"Persistent Self-Replication Table"*, is used to record the  
 data required by this operation. This Persistent Self-Replication  
 15 Table is preferably saved on a non volatile memory (typically but  
 not necessary as part of the spreadsheet disk file on the mass  
 storage 107.

Referring now to FIG. 4, the Persistent Self-Replication Table  
 20 400 corresponds to a simple logical structure made of several  
 records 401, each of which corresponds to a PSRR and includes  
 five fields:

- The *"PSRS Name"* 402 field is used for identifying uniquely the  
 PSRS associated with the current record 401.
- 25 • The *"PSRR Address"* 403 field is used for identifying uniquely  
 the PSRR within the spreadsheet. For instance, the *"PSRR  
 Address"* can correspond to the conventional address structure  
 Sheet:RowColumn..Sheet:RowColumn associated with every range

of cells (For example D:E10..D:G20 with D as Sheet name, E and G as Row name/number, 10 and 20 as Column name/number).

- The "*PSRR Pattern*" field **404** records the background pattern of the PSRR, before being member of a PSRS.
- The "*Set Index*" field **405** is used for navigating within the Self-Replication Table **400**.
- The "*Range Index*" field **406** is used for navigating within the Self-Replication Table **400**.

The record **410** located at the beginning of the Persistent Self-Replication Table **400** is referred to as the top record.

In a preferred embodiment, the Persistent Self-Replication Table **400** is explicitly included within the spreadsheet file itself, but other implementations can be used instead.

## F. Methods

### F.1 Self\_Replication\_Manager method

The method of handling user requests to take advantage of the present invention is detailed in flowchart **600** of FIGS **6A**, **6B**. This method may be thought of as the processing of the ***Self\_Replication\_Manager*** command used for creating, or deleting, or renaming a PSRS, and for adding or suppressing a PSRR to or from a PSRS. The method comprises the following steps:

- At step **601**, the method is in its default state, waiting for an event to initiate the process.
- At step **602**, an event is detected, as a result of a user action. This action may be for instance, a specific combination of keys on the keyboard **104**, or the click of the

pointing device **105** on a specific button, or any other similar means not further specified here.

- 5     • At step **603**, local variables are initialised: the **PSRSIndex** variable is set to the value 0, the **PSRRIndex** variable is set to the value 0, the **NewName** variable is set to the value "" (empty string), and the **NewRange** variable is set to the character string representing the address of the electronic spreadsheet current selection.
- 10    • At step **604**, the Self-Replication Dialog Box **510** is displayed on the display device **106**. The "Name" text box **511** is filled with the variable **NewName**. The "Existing PSRS" list box **512** is filled with the names found in the "PSRS Name" fields **402** of the various records **401** of the Self-Replicating Table **400**. Within the "Existing PSRS" list box **512**, the active item corresponds to the record **401** whose "Set Index" field **405** is equal to **PSRSIndex**. The "Existing PSRR members" list box **513** is filled with the addresses found in the "PSRR Address" fields **403** of the various records **401** of the Self-Replicating Table **400** for which the "Set Index" field **405** is equal to **PSRSIndex**. Within the "Existing PSRR members" list box **513**, the active item corresponds to the record **401** whose "Range Index" field **406** is equal to **PSRRIndex**. The "Range" text box **514** is filled with the variable **NewRange**.
- 25    • At step **605**, the method is waiting for any user action on the Self-Replication Dialog Box **510**. Such user action typically results from a click with the pointing device **105**, but may be a specific combination of keys on the keyboard **104**, or any other similar means not further specified here.
- 30    • At step **606**, a user action on the Self-Replication Dialog Box **510** is detected. If the user action is a change of the content of the "Name" text box **511**, then control is given to step **609**; if the user action is a selection with the pointing device **105**

of an item within the "Existing PSRS" list box **512**, then control is given to step **622**; if the user action is a selection with the pointing device **105** of an item within the "Existing PSRR members" list box **513**, then control is given to step **621**; if the user action is a change of the content of the "Range" text box **514**, then control is given to step **610**; if the user action is a click on the push-button "OK" **515**, then control is given to step **607**; if the user action is a click on the push-button "Cancel" **516**, then control is given to step **607**; if the user action is a click on the push-button "Create" **517**, then control is given to step **627**; if the user action is a click on the push-button "Delete" **518**, then control is given to step **626**; if the user action is a click on the push-button "Rename" **519**, then control is given to step **608**; if the user action is a click on the push-button "Add" **520**, then control is given to step **624**; if the user action is a click on the push-button "Suppress" **521**, then control is given to step **625**; if the user action is a click on the push-button "Select" **522**, then control is given to step **623**; if the user action is a click on the up arrow of the scroll bar **523**, then control is given to step **618**; if the user action is a click on the down arrow of the scroll bar **523**, then control is given to step **611**; if the user action is a click on the up arrow of the scroll bar **524**, then control is given to step **620**; if the user action is a click on the down arrow of the scroll bar **524**, then control is given to step **612**.

- At step **607**, the Self-Replication Dialog Box **510** is closed, so that it disappears from the display device **106**, and control is given back to the initial step **601** for treating any future Self\_Replication\_Manager command.

- At step **608**, in the Self-Replication Table **400**, all the records **401** having a "Set Index" field **405** equal to ***PSRSIndex*** are updated by replacing their "PSRS name" field **402** by ***NewName***. Then control is given to step **631**.
- 5 • At step **609**, the character string specified by the spreadsheet user in the "Name" text box **511** is checked against all the already defined PSRS names, as recorded in the "PSRS Name" field **402** of all the records **401** of the Self-Replication Table **400**. If the character string is new, i.e. it does not match  
10 any already defined name, then control is given to step **616**; otherwise control is given to step **613**.
- At step **610**, the character string specified by the spreadsheet user in the "Range" text box **514** is checked against a set of rules not detailed here to determine if it is or not a valid range address. Such rules are typically implementation-  
15 dependent and thus do not fall within the scope of the present invention. If the result of this checking is that this character string is found to be valid, then control is given to step **617**, otherwise control is given to step **613**.
- 20 • At step **611**, the variable ***PSRSIndex*** is decremented, unless the minimum value (equal to zero) has been reached. Then control is given to step **619**.
- At step **612**, the variable ***PSRRIndex*** is decremented, unless the minimum value (equal to zero) has been reached. Then control  
25 is given to step **631**.
- At step **613**, an error message notification is issued to warn the user that the character string checked at step **609** or **610** has not been found to be correct. This can typically be done by displaying on the display device **106** an error message in a  
30 pop-up window, but any other similar means could be used instead, without departing from the spirit of the present invention.



- At step **614**, the method is waiting for a user acknowledgement, meaning that the error message notification of step **613** has been received by the spreadsheet user.
- At step **615**, the user acknowledgement is detected. This may typically be a click, of the pointing device **105**, on a "OK" push-button within the pop-up window that is typically displayed during the step **613**, but other similar means can be used instead without departing from the spirit of the present invention. Then control is given to step **631**.
- At step **616**, the value of the variable **NewName** is set equal to the character string within the "Name" text box **511**. Then control is given to step **631**.
- At step **617**, the value of the variable **NewRange** is set equal to the character string within the "Range" text box **514**. Then control is given to step **631**.
- At step **618**, the variable **PSRSIndex** is incremented, unless the maximum value (equal to the number of different values of the "PSRS Name" field **401**) is already reached.
- At step **619**, the variable **PSRRIndex** is set equal to zero. Then control is given to step **631**.
- At step **620**, the variable **PSRRIndex** is incremented, unless the maximum value (equal to the number of records **401** whose "Set Index" field **405** is equal to **PSRSIndex**) has been reached. Then control is given to step **631**.
- At step **621**, the variable **PSRRIndex** is set equal to the value of the "Range Index" field **406** of the record **401** of the Self-Replication table **400** corresponding to the user selection of the "Existing PSRR members" list box **513**. Then control is given to step **631**.
- At step **622**, the variable **PSRSindex** is set equal to the value of the "Set Index" field **405** of the record **401** of the Self-Replication table **400** corresponding to the user selection

of the "Existing PSRS" list box **512**. Then control is given to step **631**.

- At step **623**, the method uses conventional means to let the user select a range within the spreadsheet. Such means may, for instance, rely on a pop-up window within which the user enters through the keyboard **104** the address of the range to select, or such means may rely on a pointing device **105** mode where the user clicks on the range to select, or such means may rely on other similar ways not further described here. Then control is given to step **628**.
- At step **624**, a new record **401** is added to the Self-Replication table **400**. The "PSRS Name" field **402** is set equal to the value of the "PSRS Name" field **402** of the record **401** whose "Set Index" field **405** is equal to **PSRSIndex**, the "PSRR Address" field **403** is set equal to the variable **NewRange**, the "PSRR pattern" field **404** is set equal to the value of the background pattern of the range with address **NewRange**, the "Set Index" field **405** is set equal to **PSRSIndex**, the "Range Index" field **406** is set equal to the number of PSRR incremented by one. Then control is given to step **629**.
- At step **625**, the background pattern of the range with address equal to the value of the "PSRR Address" field **403** of the record **401** whose "Set Index" field **405** is equal to **PSRSIndex** and whose "Range Index" field **406** is equal to **PSRRIndex** is set equal to the value of the "PSRR pattern" field **404** of the record **401** whose "Set Index" field **405** is equal to **PSRSIndex** and whose "Range Index" field **406** is equal to **PSRRIndex**. Then control is given to step **630**. The **Persistent\_Self\_Replicate** method is no longer set as the routine handling the cell content modification event for the range of cells with address equal to the value of the "PSRR Address" field **403** of the

record **401** whose "Set Index" field **405** is equal to **PSRSIndex** and whose "Range Index" field **406** is equal to **PSRRIndex**.

- At step **626**, all the records **401** of the Self-Replication table **400** are deleted if their "Set Index" field **405** is equal to **PSRSIndex**. Then the variable **PSRSIndex** is decremented, unless equal to zero. Then control is given to step **631**.
- At step **627**, a new record **401** is added to the Self-Replication table **400**. The "PSRS Name" field **402** is set equal to the variable **NewName**, the "PSRR Address" field **403** is left empty, the "PSRR pattern" field **404** is left empty, the "Set Index" field **405** is set equal to the number of PSRS incremented by one, and the "Range Index" field **406** is left empty. Then control is given to step **631**.
- At step **628**, the variable **NewRange** is set equal to the address of the range retrieved at step **623**. Then control is given to step **631**.
- At step **629**, the background pattern of the range of cells with address equal to **NewRange** takes the value **PATTERN\_PSRR**. The **Persistent\_Self\_Replicate** method is set as the routine handling the cell content modification event for the range of cells with address equal to **NewRange**. Then control is given to step **631**.
- At step **630**, the record **401** in the Self-Replication table **400** with "Set Index" field **405** equal to **PSRSindex** and with "Range Index" field **406** equal to **PSRRindex**, is deleted. Then the variable **PSRRIndex** is decremented, unless equal to zero.
- At step **631**, the Self-Replication Table **400** is first rearranged and then sorted so that the set of values taken by the "Set Index" fields **405** is contiguous and so that the sets of values taken by the "Range Index" field **406** for a given value of the "Set Index" field **405** are contiguous. Then the

Self-replication Dialog Box **510** display fields are refreshed. The "Name" text box **511** is filled with the variable **NewName**. The "Existing PSRS" list box **512** is filled with the names found in the "PSRS Name" fields **402** of the various records **401** of the Self-Replicating Table **400**. Within the "Existing PSRS" list box **512**, the active item corresponds to the record **401** whose "Set Index" field **405** is equal to **PSRSIndex**. The "Existing PSRR members" list box **513** is filled with the addresses found in the "PSRR Address" fields **403** of the various records **401** of the Self-Replicating Table **400** for which the "Set Index" field **405** is equal to **PSRSIndex**. Within the "Existing PSRR members" list box **513**, the active item corresponds to the record **401** whose "Range Index" field **406** is equal to **PSRRIndex**. The "Range" text box **514** is filled with the variable **NewRange**. Then control is given back to step **605** for waiting for a new user action to treat.

### F.3. Persistent\_Self\_Replicate method

The method for automatically reflecting an update of the content of a PSRR onto the other PSRR belonging to the same PSRS to take advantage of the present invention is summarized in flowchart **700** of FIG **7**. This method may be thought of as the processing of the "**Persistent\_Self\_Replicate**" command which is invoked each time the content of a PSRR is changed, as outlined in the step **629** of the **Self\_Replication\_Manager** method. The method comprises the following steps:

- At step **701**, the method is in its default state, waiting for an event to initiate the process.

- At step **702**, an event is detected, as a result of a PSRR content update.
- At step **703**, the address of the updated PSRR, considered as a parameter of the **Persistent\_Self\_Replicate** command, is retrieved under the name CurrPSRR.
- At step **704**, a regular copy operation is performed on the PSRR with address CurrPSRR.
- At step **705**, the top record **410** of the Self-Replication Table **400** is set as the current record **401** of the table.
- At step **706**, the "PSRR Address" field **403** of the current record **401** of the Self-Replication Table **400** is compared against CurrPSRR. If found equal, then control is given to step **707** ; otherwise control is given to step **713**.
- At step **707**, the local variable CurrSetIndex is set equal to the value of the "Set Index" field **405** of the current record **401** of the Self-Replication Table **400**.
- At step **708**, the top record **410** of the Self-Replication Table **400** is set as the current record **401** of the table.
- At step **709**, the "Set Index" field **405** of the current record **401** of the Self-Replication Table **400** is compared against CurrSetIndex. If found equal, then control is given to step **710** ; otherwise control is given to step **711**.
- At step **710**, a regular paste operation is performed on the range of cells pointed by the "PSRR Address" field **403** of the current record **401** of the Self-Replication Table **400**.
- At step **711**, a test is performed to check if the current record **401** of the Persistent Copy-Paste Table **400** is in fact the last record of this table. If it is the last record, then control is given to the initial step **701** for handling any new future command; otherwise control is given to step **712**.

- At step **712**, the next record of the Persistent Copy-Paste Table **400** is set as the new current record **401** of this table. Then control is given to step **709**.
- At step **713**, a test is performed to check if the current record **401** of the Self-Replication Table **400** is in fact the last record of this table. If it is the last record, then control is given to step **714**; otherwise control is given to step **715**.
- At step **714**, a "Should Not Occur" condition is logged as it is normally impossible not to find in the Self-Replication table **400** a record **401** with a "PSRR Address" field **403** equal to the parameter CurrPSRR of the command. Then control is given to step **701** for handling any new future command.
- At step **715**, the next record of the Self-Replication Table **400** is set as the new current record **401** of this table. Then control is given to step **706**.

#### ALTERNATE EMBODIMENTS

While the invention has been particularly shown and described with reference to a preferred embodiment, it will be understood that various changes in form and detail may be made therein without departing from the spirit, and scope of the invention.

The Persistently Self-Replication method and system according to the present invention may be used advantageously in those environments where elements of information are organised as multidimensional tables having more than three dimensions.